# Tìm hiểu về ES6

## ES6 là gì

**ES6** là chữ viết tắt của **ECMASript 6** được ra đơi vào năm 2015, đây được coi là một tập hợp các kỹ thuật nâng cao của Javascript.

## Các chức năng của ES6

**1. Var, let, const**

***var***

Trước ES6 ra đời thì từ khóa ***var*** là từ khóa mặc định dùng để khai báo đa dạng các loại biến như string, boolean, enum, … Biến được khai báo bằng ***var*** sẽ có phạm vi là *globally scoped*.

|  |  |  |
| --- | --- | --- |
| **VD:**   |  | | --- | | var fullName = “Thanh Huyen”;  function showFullName() {  if(fullName === “Thanh Huyen”) {  fullName= “Le Thanh Huyen”  }  console.log(fullName);  }  console.log(fullName);  showFullName(); |   **Kết quả:**   |  | | --- | | “Thanh Huyen”  “Le Thanh Huyen” | |

Như vậy. việc khai báo biến bằng từ khóa ***var*** khiến cho biến đó sẽ có hiệu lực ở mọi nơi.Ngoài ra, ta có thể thay đổi giá trị của biến đó được.

Thêm vào đó, ta có thể khai báo lại biến ***var***. Ta xét ví dụ sau:

|  |  |  |
| --- | --- | --- |
| **VD:**   |  | | --- | | var fullName = “Thanh Huyen”;  var time = 3;  if(time <4) {  var fullName = “Le Thanh Huyen”;  }  console.log(fullName); |   **Kết quả**   |  | | --- | | “Le Thanh Huyen” | |

Vì thỏa mãn điều kiện ***if*** nên ***fullName*** được thay đổi. Việc này sẽ gây ra khó khăn khi chương trình có lỗi và phải fix bug, ta không biết giá trị của biến có bị thay đổi hay không. Vì cậy ES6 cung cấp hai cách khai báo biến khác là ***let*** và ***const***

***let***

Nguyên nhân khiến ***let*** có thể giải quyết vấn đề mà ***var*** gặp phải là do let có phạm vi là *block scoped*, và khi biến được khai báo bằng ***let*** thì nó chỉ có thể thay đổi giá trị chứ không được khai báo lại.

|  |  |
| --- | --- |
| **VD:**   |  | | --- | | let fullName = “Thanh Huyen”;  let fullName = “Le Thanh Huyen”; //error :identifier “fullName” has already been declared | |

Tuy nhiên với các block khác nhau thì việc khai báo biến sẽ không sinh ra lỗi vì đối với từng scope, mỗi biến được xem xét là một biến riêng biệt

|  |  |  |
| --- | --- | --- |
| **VD:**   |  | | --- | | let fullName = “Thanh Huyen”;  if(true) {  let fullName = “Le Thanh Huyen”;  console.log(fullName);  }  console.log(fullName); |   **Kết quả**   |  | | --- | | “Le Thanh Huyen”  “Thanh Huyen” | |

***const***

Tương tự với ***let*** thì biến được khai báo bằng từ khóa ***const*** sẽ có phạm vi là *block scoped*. Nếu trường hợp kiểu của biến là primitive (bao gồm string, number, Boolean, null và undefined) thì chúng không thể tái khai báo hay cập nhật giá trị mới thay thế cho giá trị trước đó

|  |  |
| --- | --- |
| **VD:**   |  | | --- | | const fullName = “Thanh Huyen”;  fullName = “Le Thanh Huyen”;//error: assignment to constant variable  const fullName = “Le Thanh Huyen”; //error: identifier “fullName” has already been declared | |

Đối với trường hợp kiểu biến là reference (bao gồm object, array và function) thì tuy không khai báo hay cập nhật giá trị của biến được nhưng chúng ta vẫn có thể cập nhật giá trị thuộc tính của biến đó

|  |  |  |
| --- | --- | --- |
| **VD:**   |  | | --- | | const fullName = {  firstName : “Huyen”,  lastName : “Thanh”  }  fullName.lastName = “Le Thanh”;  console.log(fullName); |   **Kết quả**   |  | | --- | | {firstName : “Huyen”, lastName : “Le Thanh”} | |

**Hoisting**

Là cơ chế của JS cho phép các khai báo biến và hàm được dời lên đầu của scope của chúng trước khi thực thi đoạn code.

Điều này có nghĩa là, không thành vấn đề việc biến và hàm được khai báo ở đâu, chúng sẽ được dời lên trên đầu của scope của chúng bất chấp scope của chúng là toàn cục (global) hay cục bộ (local). Và lưu ý, là Javascirpt chỉ dời phần khai báo, còn việc gán giá trị vẫn giữ nguyên vị trí.

Ta xét ví dụ sau

|  |  |
| --- | --- |
| **VD:**   |  | | --- | | console.log(fullName);  var fullName = “Thanh Huyen”; | |

Kết quả sẽ ra undefined do trình biên dịch sẽ hiểu như sau

|  |  |
| --- | --- |
| |  | | --- | | var fullName;  console.log(fullName);  var fullName = “Thanh Huyen”; | |

***let*** cũng có tính *hoisting* tuy nhiên, nếu ***var*** được khởi tạo với giá trị là undefined thì ***let*** sẽ không có bất kỳ giá trị khởi tạo nào.

## 2. arrow function

Đây là cú pháp ngắn gọn hơn dùng để viết function. ***Arrow function*** là một hàm vô danh và nó là thay đổi this blind đến function. Việc dùng ***arrow function*** giúp ta tránh phải gõ từ khóa *function*, *return* và dấu ngoặc nhọn

***Trong trường hợp nhiều tham số***

*Cú pháp:*

**(param1, param2, …) => expression**

|  |  |  |
| --- | --- | --- |
| **VD:**   |  | | --- | | var multiply = function (x, y) {  return x\*y;  } |   **Sẽ được thay thế bằng:**   |  | | --- | | var multiply = (x, y) => x\* y; | |

***Trong trường hợp không có tham số***

*Cú pháp*

**() => expression**

|  |  |  |
| --- | --- | --- |
| **VD:**   |  | | --- | | var showFullName = function () {  console.log(fullName);  } |   **Được thay thế bằng:**   |  | | --- | | var showFullName = () => {console.log(fullName);} | |

**this**

***this*** dùng để trỏ tới chính object gọi hàm đó

|  |  |  |
| --- | --- | --- |
| **VD:**   |  | | --- | | const fullName = {  firstName : “Huyen”,  lastName : “Thanh”,  getFullName : function () {  return this.firstName + this.lastName  }  }  fullName.getFullName() |   **Kết quả:**   |  | | --- | | “Huyen Le” | |

Ta xét ví dụ sau:

|  |  |  |
| --- | --- | --- |
| **VD:**   |  | | --- | | const fullName = {  firstName : “Huyen”,  lastName : “Thanh”,  getFullName : function () {  return this.firstName + this.lastName  }  }  fullName.getFullName()  const getName = fullName.getFullName()  getName() // error : this trỏ tới object của window |   **Cách sửa:**   |  | | --- | | const getName = fullName.getFullName.blind(fullName) | |

Sau khi dung ***blind*** thì con trỏ sẽ trỏ tới ***fullName***

Việc sử dụng ***blind*** sẽ tránh được tình trạng “*mập mờ context*”

Khác với function thông thường, ***arrow function*** không có ***blind***, vì vậy, nó không có từ khóa ***this***. Chính vì vậy thì this sẽ tương ứng với ngữ cảnh gần nhất của nó.

Cũng vì nó không có định nghĩa this nên ***arrow function*** không phù hợp là method của object.